Juris Hartmanis

Turing Award Lecture

On Computational Complexity and the Nature of Computer Science

In scientific work, the recognition by one's peers is one of the greatest rewards. In particular, an official recognition by the scientific community, as Richard Stearns and I are honored by the 1993 ACM Turing Award, is very satisfying and deeply appreciated.[1] Science is a great intellectual adventure and one of humankind's greatest achievements. Furthermore, a research career can be an exciting, rewarding and ennobling activity, particularly so if one is fortunate to participate in the creation of a completely new and very important science, as many scientists are. My road to computer science was not a direct one. Actually it looks more like a random walk, in retrospect, with the right intellectual stops to prepare me for work in computer science.

I was born in Latvia, which lost its independence during World War II and from which we had to flee because of heavy fighting at the end of World War II. After the war as a D.P. (displaced person) in Germany, I finished a superb Latvian high school in a D.P. camp staffed by elite refugee academics who conveyed their enthusiasm for knowledge, scholarship and particularly for science. I studied physics at the Philips University in Marburg and waited for a chance to emigrate to the United States. This chance came after about two-and-a-half years of studies. In the U.S. our sponsors were in Kansas City, and, after arriving there, I proceeded to the University of Kansas City (now part of the University of Missouri system). My two-plus years of study were judged to be the equivalent of a bachelor's degree, and I was accepted for graduate work and very generously awarded a fellowship. Since there was no graduate program in physics, I was advised (or told) to study mathematics, which had a graduate program. A year later I emerged with a master's degree in mathematics and with a far better appreciation of the power and beauty of mathematics. The California Institute of Technology accepted me for graduate work and from my record decided that I looked like "an applied mathematician" (which is probably what you get if you mix two years of European physics with a year of Kansas City mathematics, though I had never taken a course in applied mathematics). Since there was at that time no program in applied mathematics at Cal Tech, I was advised I would be perfectly happy studying pure mathematics.

This was good advice, and four years later, after one of the most stimulating intellectual periods in my life, I had earned my Ph.D. in mathematics with a dissertation in lattice theory and a minor in physics. Though I loved pure mathematics and was impressed by the beauty and power of mathematical abstractions, I felt some intellectual restlessness and a hope to find research problems with a more direct link to the world around us. Still, I followed my advisor's recommendation and accepted a faculty position in mathematics at Cornell University. During the second year at Cornell I was offered and accepted a summer job at General Electric Research Laboratory in Schenectady, N.Y., in their new information studies section headed by Dr. Richard Shuey. That summer was a sharp turning point in my scientific interests. At the GE Research Laboratory, I was caught up in the excitement about participating in the creation of a new science about information and computing. Computer science offered me the hoped-for research area with the right motivation, scope and excitement. One academic year later I joined the GE Research Laboratory as a research scientist. The following year Richard Stearns, a mathematics graduate student at Princeton, spent a summer at the Laboratory where we started our collaboration. After completing his Ph.D. at Princeton with a dissertation in game theory, Dick joined the Laboratory and we intensified our collaboration.

Our views of what kind of computer science we wanted to do were influenced by our backgrounds and the intensive study of the relevant literature we could find. We delighted in Turing's 1936 paper [14] and were impressed by the elegance, crispness and simplicity of the undecidability results and basic recursive function theory. Turing's work supplied us with the necessary well-defined abstract computer model in our later work. I personally was deeply impressed with Shannon's communication theory [12]. Shannon's theory gave precise quantitative laws of how much information can be "reliably" transmitted over a noisy channel in terms of the channel capacity and the entropy of the information source. I loved physics for its beautifully precise laws that govern and explain the behavior of the physical world. In Shannon's work, for the first time, I saw precise quantitative laws that governed the behavior of the abstract entity of information. For an ex-physicist the idea that there could be quantitative laws governing such abstract entities as information and its transmission was surprising and immensely fascinating. Shannon had given a beautiful example of quantitative laws for information which by its nature is not directly constrained by physical laws. This raised the question whether there could be precise quantitative laws that govern the abstract process of computing, which again was not directly constrained by physical laws. Could there be quantitative laws that determine for each problem how much computing effort (work) is required for its solution and how to measure and determine it?

From these and other considerations grew our deep conviction that there must be quantitative laws that govern the behavior of information and computing. The results of this research effort were summarized in our first paper on this topic, which also named this new research area, "On the computational complexity of algorithms" [5]. To capture the quantitative behavior of the computing effort and to classify computations by their intrinsic computational complexity, which we were seeking, we needed a robust computing model and an intuitively satisfying classification of the complexity of problems. The Turing machine was ideally suited for the computer model, and we modified it to the multi-tape version. To classify computations (or problems) we introduced the key concept of a complexity class in terms of the Turing machines with bounded computational resources. A complexity class, for example, C_{n^2} in our original notation, consists of all problems whose instances of length n can be solved in n^2 steps on a multi-tape Turing machine. In contemporary notation, $C_{n^2} = \mathrm{TIME}[n^2]$.

Today, complexity classes are central objects of study, and many results and problems in complexity theory are expressed in terms of complexity classes.

A considerable part of our early work on complexity theory was dedicated to showing that we had defined a meaningful classification of problems according to their computational difficulty and deriving results about it. We showed that our classification was robust and was not essentially altered by minor changes in the model and that the complexity classification indeed captured the intuitive ideas about the complexity of numbers and functions. We explored how computation speed changed by going from one-tape to multi-tape machines and even to multi-dimensional tapes and derived bounds for these "speed-

[1]The Turing Award Lecture by co-recipient Richard Stearns will appear in the November issue of *Communications of the ACM*.

ups." Somewhat later, Manuel Blum, in his Ph.D. dissertation at MIT [1], developed an axiomatic theory of computational complexity and, among many other results, showed that all complexity measures are recursively related. Our speed-up results were special cases of this relationship. For us it was a delight to meet Manuel while he was writing his dissertation and to exchange ideas about computational complexity. Similarly, we were impressed and influenced by H. Yamada's work on real-time computations in his dissertation at the University of Pennsylvania [15] under the supervision of Robert McNaughton. We also proved Hierarchy Theorems that asserted that a slight increase in computation time (bounds) permits solution of new problems. More explicity: if $T(n)$ and $U(n)$ are "nice" functions and

$$\lim_{n->\infty} \frac{T(n)^2}{U(n)} = 0$$

then complexity class TIME[$T(n)$] is properly contained in TIME[$U(n)$]. These results showed that there are problems with very sharp, intrinsic computational complexity bounds. No matter what method and computational algorithm was used, the problem solution required, say n^2, operation for problem instance of size n. Blum in his dissertation showed that this is not the case for all problems and that there can exist exotic problems with less sharply defined bounds.

To relate our classification of the real numbers by their computation complexity to the classical concepts, we showed that all algebraic numbers are in the low complexity class TIME[n^2] and found, to our surprise, some transcendental numbers that were real-time computable (i.e., they were in TIME[n]). Since we could not prove that any irrational algebraic numbers were in TIME[n], we conjectured that all real-time computable numbers are either rational or transcendental. This is still an open problem 30 years later and only gradually did we realize its mathematical depth and the profound consequences its proof would have in mathematics.

Toward the end of the introduction of our first paper on complexity theory [5], we state: "The final section is devoted to open questions and problem areas. It is our conviction that numbers and functions have an intrinsic computational nature according to which they can be classified, as shown in this paper, and that there is a good opportunity here for further research." Indeed there was! We had opened a new computer science area of research and given it a name.

At the GE Research Laboratory, Phil Louis joined us to explore tape- (or memory-) bounded computations that yielded many interesting results and established computational space as another major computational resource measure [7, 13]. We showed that all context-free languages could be recognized on $(\log n)^2$-tape. This result led Savitch [10] to his elegant result about the relation between deterministic and nondeterministic tape-bounded computations: for "nice" functions $F(n)$, NTAPE[$F(n)$] is contained in TAPE[$F(n)^2$].

Our colleague at the Laboratory, Daniel Younger [16], showed that context-free languages were contained in TIME[n^3]. Soon many others joined the exploration of the complexity of computation, and computational complexity theory grew into a major research area with deep and interesting results and some of the most notorious open problems in computer science.

Looking at all of computer science and its history, I am very impressed by the scientific and technological achievements, and they far exceed what I had expected. Computer science has grown into an important science with rich intellectual achievements, an impressive arsenal of practical results and exciting future challenges. Equally impressive are the unprecedented technological developments in computing power and communication capacity that have amplified the scientific achievements and have given computing and computer science a central role in our scientific, intellectual and commercial activities.

I personally believe that computer science is not only a rapidly maturing science, but that it is more. Computer science differs so basically from the other sciences that it has to be viewed as a new species among the sciences, and it must be so understood. Computer science deals with information, its creation and processing, and with the systems that perform it, much of which is not directly restrained and governed by physical laws. Thus computer science is laying the foundations and developing the research paradigms and scientific methods for the exploration of the world of information and intellectual processes that are not directly governed by physical laws. This is what sets it apart from the other sciences and what we vaguely perceived and found fascinating in our early exploration of computational complexity.

One of the defining characteristics of computer science is the immense difference in scale of the phenomena computer science deals with. From the individual bits of programs and data in the computers to billions of operations per second on this information by the highly complex machines, their operating systems and the various languages in which the problems are described, the scale changes through many orders of magnitude. Donald Knuth[2] puts it nicely:

Computer Science and Engineering is a field that attracts a different kind of thinker. I believe that one who is a natural computer scientist thinks algorithmically. Such people are especially good at dealing with situations where different rules apply in different cases; they are individuals who can rapidly change levels of abstraction, simultaneously seeing things "in the large" and "in the small."

The computer scientist has to create many levels of abstractions to deal with these problems. One has to create intellectual tools to conceive, design, control, program, and reason about the most complicated of human creations. Furthermore, this has to be done with unprecedented precision. The underlying hardware that executes the computations are universal machines and therefore they are chaotic systems: the slightest change in their instructions or data can result in arbitrarily large differences in the results. This, as we well know, is an inherent prop-

[2]Personal communication. March 10, 1992 letter.

erty of universal computing devices (and theory makes clear that giving up universality imposes a very high price). Thus computer scientists are blessed with a universal device which can be instructed to perform any computation and simulate in principle any physical process (as described by our current laws of physics), but which is therefore chaotic and must be controlled with unprecedented precision. This is achieved by the successive layers of implemented abstraction wrapped around the chaotic universal machines that help to bridge the many orders of magnitude in the scale of things.

It is also this universality of the computing devices that gives the computing paradigm its immense power and scope. During various periods people have used the conceptualizations of their newest devices to try to understand and explain how nature and humans function. Thus our current heavy reliance on computer concepts and computer simulations for various phenomena has been compared to the use of the explanatory role of steam-driven devices, gears and latches, or clocks.

The universality of digital computers and the ever-increasing computing power give the computing paradigm a different and a very central role in all of our intellectual activities. The digital computer is a universal device and can perform in principle any computation (assuming the Church-Turing thesis, it captures all computations) and, in particular, any mathematical procedure in an axiomatized formal system. Thus in principle the full power of mathematical reasoning, which has been civilization's primary scientific tool, can be embodied in our computers from numerical computations and simulation of physical processes to symbolic computations and logical reasoning to theorem proving. This universality and the power of modern computers are indeed very encompassing of our intellectual activities and growing in scope and power.

Clearly, computer science is not a physical science; still, very often it is assumed that it will show strong similarities to physical sciences and may have similar research paradigms in regard to theory and experiments. The failure of computer science to conform to the paradigms of physical sciences is often interpreted as immaturity of computer science. This is not the case, since theory and experiments in computer science play a different role than in physical sciences. For a more detailed contrasting of the research paradigms in physics and computer science, see [4].

Even a brief look at research topics in computer science reveals the new relation between theory and experiments. For example, the design and analysis of algorithms is a central theme in theoretical computer science. Methods are developed for their design, measures are defined for various computational resources, trade-offs between different resources are explored, and upper- and lower-resource bounds are proved for the solutions of various problems. Similarly, theory creates methodologies, logics and various semantic models to help design programs, to reason about programs, to prove their correctness, and to guide the design of new programming languages. Theories develop models, measures and methods to explore and optimize VLSI designs, and to try to conceptualize techniques to design efficient computer and communications systems.

Thinking about the previously mentioned (and other) theoretical work in computer science, one is led to the very clear conclusion that theories do not compete with each other for which better explains the fundamental nature of information. Nor are new theories developed to reconcile theory with experimental results that reveal unexplained anomalies or new, unexpected phenomena as in physics. In computer science there is no history of critical experiments that decide between the validity of various theories, as there are in physical sciences.

The basic, underlying mathematical model of digital computing is not seriously challenged by theory or experiments. The ultimate limits of *effective* computing, imposed by the theory of computing, are well understood and accepted. There is a strong effort to define and prove the *feasible* limits of computation, but even here the basic model of computation is not questioned. The key effort is to prove that certain computations cannot be done in given resource bounds, well illustrated by the $P = NP$? question. One should note that the solution of this problem could have broad implications. For example, it could give proof of what encryption procedures are safe under what attacks and for how long. It could also lead to a deeper understanding of the limits of human-computer reasoning power. In general, the "separation" problems, that is the questions if $P \neq NP \neq$ PSPACE $\neq$ EXPTIME $\neq$ NEXPTIME $\neq$ EXPSPACE? are among the most important open problems in theoretical computer science. But there are no experiments, physical or computational, which could resolve these problems, again emphasizing the different scientific nature of computer science.

In computer science, results of theory are judged by the insights they reveal about the mathematical nature of various models of computing and/or by their utility to the practice of computing and their ease of applicability. Do the models conceptualize and capture the aspects computer scientists are interested in, do they yield insights in design problems, do they aid reasoning and communication about relevant problems? In the design and analysis of algorithms, which is a central theme in theoretical computer science, the measures of performance are well defined, and results can be compared quite easily in some of these measures (which may or may not fully reflect their performance on typical problems). Experiments with algorithms are used to test implementations and compare their "practical" performance on the subsets of problems deemed important.

Similarly, an inspection of the experimental work and systems building in computer science reveals a different pattern than in physical sciences. Such work deals with performance measurements, evaluation of design methodologies, testing of new architectures, and above all, testing feasibility by building systems to do what has never been done before.

Systems building, hardware and software, is the defining characteristic of applied and/or experimental work in computer science (though experimental is not meant in the old sense). This has the consequence that computer

Looking at all of computer science and its history,

I am very impressed by the scientific and technological achievements,

and they far exceed what I had expected.

science advances are often demonstrated and documented by a dramatic demonstration rather than a dramatic experiment as in physical sciences. It is the role of the demo to show the possibility or feasibility to do what was thought to be impossible or not feasible. It is often that the (ideas and concepts tested in the) dramatic demos influence the research agenda in computer science.

This is reflected in the battle cry of the young computer scientists, "demo or die," which is starting to rival the older "publish or perish," which is still valid advice, but should be replaced by "publish in refereed journals or perish."

From the preceding observations we can see that theory and experiments in computer science are contributing to the design of algorithms and computing systems that execute them, that computer science is concentrating more on the *how* than the *what,* which is more the focal point of physical sciences. In general the *how* is associated with engineering, but computer science is not a subfield of engineering. Computer science is indeed an independent new science, but it is intertwined and permeated with engineering concerns and considerations. In many ways, the science and engineering aspects in computer science are much closer than in many other disciplines. To quote Fred Brooks [2] about programming:

The programmer, like the poet, works only slightly removed from pure thought-stuff. He builds his castles in the air, from air, creating by exertion of the imagination. Few media of creation are so flexible, so easy to polish and re-work, so readily capable of realizing grand conceptual structures. (. . . later, this very tractability has its own problems.)

Yet the program construct, unlike the poet's words, is real in the sense that it moves and works, producing visible outputs separate from the construct itself. It prints results, draws pictures, produces sounds, moves arms. The magic of myth and legend has come true in our time. One types the correct incantation on a keyboard, and a display screen comes to life, showing things that never were nor could be.

Webster's dictionary defines engineering as "the application of scientific principles to practical ends as the design, construction, and operation of efficient and economical structures, equipment and systems." By this definition, much of computer science activity can be viewed as engineering or at least the search for those scientific principles which can be applied "to practical ends, design, construction, . . . " But again, keeping in mind Brooks' quote and reflecting on the scope of computer science and engineering activities, we see that the engineering in our field has different characteristics than the more classical practice of engineering. Many of the engineering problems in computer science are not constrained by physical laws, and they demand the creation of new engineering paradigms and methodology.

As observed earlier, computer science work is permeated by concepts of efficiency and search for optimality. The "how" motivation of computer science brings engineering concepts into the science, and we should take pride in this nearness of our science to applicability.

Somewhat facetiously, but with a grain of truth in it, we can say that computer science is the engineering of mathematics (or mathematical processes). In these terms we see very strongly that it is a new form of engineering.

I am deeply convinced that we should not try to draw a sharp line between computer science and engineering and that any attempt to separate them is counterproductive.

At the same time, I am convinced that computer science already has made and has a tremendous potential to make contributions to the understanding of our physical and intellectual world. The computing paradigm, supported by ever more powerful universal computing devices, motivates and permits the exploration and simulation of physical and intellectual processes and even assesses their power and limitations.

Already Warren McCullach in 1964 [8] had a vision of "Experimental epistemology, the study how knowledge is embodied in the brains and may be embodied in machines." John McCarthy [9] states less modestly, "The study of AI may lead to a mathematical metaepistemological analogous to metamathematics—to a study of the relations between a knower's rule for accepting evidence and the world in which he is embedded. This study could result in mathematical theorems about whether certain intellectual strategies can lead to the discovery of certain facts about the world. I think that this possibility will eventually revolutionize philosophy."

The computing paradigm has permitted the clarification of the concept of randomness by means of Kolmogorov complexity of finite and infinite sequences [6]. Again, the universality of the computing model was essential to prove the validity of these concepts.

Computational complexity considerations have refined the concepts of randomness relative to the intended applications. It has been shown that what is (acceptable or passes as) random depends on the computing power in the application. Still there remain deep open problems in this area about physical processes and computing. The Kolmogorov random strings are not computable, in a very strong sense; no Turing machine can print a Kolmogorov random string longer than its size (description). Does a

corresponding law (theorem?) hold for all physical systems? Can small physical systems produce long Kolmogorov random strings, or better yet, can a finite physical system (properly formulated with the needed energy inflow without adding randomness?) produce unbounded Kolmogorov random sequence? If so, then indeed physical processes are not fully capturable by computer simulation.

Very recently, computer science motivation has led to the study of interactive proofs and proof checking, revealing unexpected power of randomization and interaction between prover and verifier [11]. These results show that with very few questions about a long proof a verifier can be convinced with arbitrarily high probability that there is a correct proof without ever seeing the whole proof. These and related results have given fundamental new insights about the nature of mathematical proofs and are indeed metamathematical results.

Recursive function theory, originally motivated by Goedel's incompleteness results, classified what is and is not *effectively* computable, thus clearly showing the power and limitations of formal mathematical reasoning. Complexity theory is currently struggling to determine what is and is not *feasibly* computable. In this effort the $P = NP$? problem is the best known open problem in this struggle, but by far not the only open separation problem. When the $P = NP$? and other separation problems are resolved and deeper insights are gained about the limits of the feasibly computable, the computing paradigm and complexity theory may allow us to better understand the power and limitations of the human-computer reasoning. I believe that computer science has the potential to give deep new insights and quantitative understanding of the computing paradigm and our intellectual processes and thus, just maybe, a possibility to grasp the limits of the knowable.

Acknowledgments

The views expressed here have been deeply influenced by the author's participation in the National Research Council study resulting in the report, *Computing the Future: A Broader Agenda for Computer Science and Engineering* [3] and by discussions with colleagues at Cornell University and at the Max Planck Institut fur Informatik in Saarbruecken, Germany. Particularly influential have been Robert Constable, Fred Schneider, and Richard Zippel. The importance of demos in computer science was emphasized by Constable and the importance of the immense differences in the scale of phenomena in computer science was eloquently explained by Zippel and compared to the still badly understood phenomena of turbulence in fluid dynamics, where the wide range of scales is contributing to the difficulty of the problem. ◘

References

1. Blum, M. A machine independent theory of the complexity of recursive functions. *J. ACM 14* (1967), 322–336.
2. Brooks, F.P. Jr. *The Mythical Man-Month: Essays on Software Engineering*. Addison-Wesley, Reading, Mass., 1975.
3. Hartmanis, J. and Lin, H., Eds. *Computing the Future: A Broader Agenda for Computer Science and Engineering*. National Academy Press, Washington, D.C., 1992.
4. Hartmanis, J. Some observations about the nature of com-

puter science. In *Foundations of Software Technology and Theoretical Computer Science*. Lecture Notes in Computer Science, Vol. 761. Springer-Verlag, 1993, 1–12.
5. Hartmanis, J. and Stearns, R.E. On the computational complexity of algorithms. *Trans. Amer. Math. Soc.*, 177 (1965), 285–306.
6. Li, M. and Vitanyi, P.M.B. *An Introduction to Kolmogorov Complexity and Its Applications*. Springer-Verlag, Heidelberg, Germany, 1993.
7. Lewis, P.M., Stearns, R.E., and Hartmanis, J. Memory bounds for the recognition for context-free and context-sensitive languages. In *Proceedings of IEEE Sixth Annual Symposium on Switching Circuit Theory and Logical Design*. (1965), pp. 191–202.
8. McCullach, W.S. A historical introduction to the postulational foundations of experimental epistemology. In *Cross-Cultural Understanding: Epistemology in Anthropology*. F.C.S. Northrop, and H.H. Livingston, Eds., Harper and Row, New York, 1964.
9. McCarthy, J. Mathematical logic and artificial intelligence. In *The Artifical Intelligence Debate*. S.R. Graubard, Ed., MIT Press, Cambridge, Mass., 1988.
10. Savitch, W.J. Relationship between nondeterministic and deterministic tape complexities. *J. Comput. Syst. Sci.*, 4 (1970), 177–192.
11. Shamir, A. IP = PSPACE. *J. ACM 39* (1992), 869–877.
12. Shannon, C. The mathematical theory communication. *Bell System Tech. J. 27* (1948), 379–656.
13. Stearns, R.E., Hartmanis, J., and Lewis, P.M. Hierarchies of memory limited computations. In *Proceedings of IEEE Sixth Annual Symposium on Switching Circuit Theory and Logical Design*. (1965), pp. 179–190.
14. Turing, A.M. On computable numbers with an application to the Entscheidungaproblem. In *Proceedings of the London Mathematical Society*, series 2, 42 (1936), 230–265.
15. Yamada, H. Real-time computation and recursive functions not real-time computable, *IEEE Trans. Elec. Comput. 11*, 6 (1962), 753–760.
16. Younger, D.H. Recognition and parsing of context-free languages in time n^3. *Information and Control 10*, 2 (1967), 189–208.

About the Author:
JURIS HARTMANIS is the Walter R. Read Professor of Engineering in the department of computer science at Cornell University. Current research interests include theory of computing, computational complexity, and structural complexity. **Author's Present Address:** Department of Computer Science, Cornell University, 5149 Upson Hall, Ithaca, NY 14853; email: jh@cs.cornell.edu

This research was supported in part by National Science Foundation grant #CCR-9123730 and by the Alexander von Humboldt Foundation and the Max Planck Institut fur Informatik in Saarbruecken, Germany.